Multi-Robot Control (MRC)

W3: ROS and MATLAB/Simulink
roslaunch and rosbag
MATLAB and ROS
Plan: ROS and MATLAB/Simulink

Last Week
- Parker, “Distributed Intelligence...”
- ROS:
  - Packages
  - Nodes, topics and services (publish/subscribe)
  - Tools
    - rosnodex list/info
    - rostopic list/pub/info/echo
    - rosservice list/info/call
    - rqt_graph

This Week
- Mataric, “Behavior-Based Control”
- ROS
  - Launch files and remap
  - Recording and playing logs (bag files)
- MATLAB/Simulink
  - Processing bag files
  - Timing
  - Waypoint guidance
What happens when we have lots of nodes, remapping, configuration, etc.?

- Opening N terminal windows get annoying
- Launch files are recipes for more complex operations
  - Repeatable
  - Distributable

Documented XML format for scripting ROS
- Automatically instantiates roscore if necessary
```xml
<?xml version="1.0"?>

<!-- Simple example -->

<launch>
  <node pkg="turtlesim" type="turtlesim_node" name="my_simulated_turtle"/>
</launch>
```

```
$ roslaunch mrc_hw3 turtle_ex.launch
```
<launch>
  <arg name="port" default="$(optenv HUSKY_PORT /dev/prolific)" /> 

  <node pkg="clearpath_base" type="kinematic_node" name="husky_kinematic" ns="husky">
    <param name="port" value="$(arg port)" />
    <rosparam>
      cmd_fill: True
      data:
        system_status: 10
        safety_status: 10
        encoders: 10
        differential_speed: 10
        differential_output: 10
        power_status: 1
    </rosparam>
  </node>

  <!-- Publish diagnostics information from low-level MCU outputs -->
  <node pkg="husky_base" name="husky_base_diagnostics" type="diagnostics_publisher" />

  <!-- Publish wheel odometry from MCU encoder data -->
  <node pkg="husky_base" name="husky_basic_odom" type="basic_odom_publisher" />

  <!-- Diagnostic Aggregator -->
  <node pkg="diagnostic_aggregator" type="aggregator_node" name="diagnostic_aggregator">
    <rosparam command="load" file="$(find husky_base)/config/diagnostics.yaml"/>
  </node>
</launch>
<launch>
    <node pkg="turtlesim" type="turtlesim_node" name="simulated_turtle">
        <remap from="turtle1/cmd_vel" to="XXXXXX" />
    </node>
</launch>
rosbag

Collection of tools for logging and replaying ROS data

Indispensable part of middleware

• Documenting
  – Did it do what it was supposed to?
  – Writing the thesis!

• Debugging
  – Why didn't it do what it was supposed to?

• Development
  – Running a new robot in an old situation
  – Standard data sets
  – Training data
bsb@aku:~$ rosbag -h
Usage: rosbag <subcommand> [options] [args]

A bag is a file format in ROS for storing ROS message data. The rosbag command can record, replay and manipulate bags.

Available subcommands:
  check               Determine whether a bag is playable in the current system, or if it can be migrated.
  compress            Compress one or more bag files.
  decompress           Decompress one or more bag files.
  filter               Filter the contents of the bag.
  fix                  Repair the messages in a bag file so that it can be played in the current system.
  help
  info                 Summarize the contents of one or more bag files.
  play                 Play back the contents of one or more bag files in a time-synchronized fashion.
  record              Record a bag file with the contents of specified topics.
  reindex             Reindexes one or more bag files.

For additional information, see http://wiki.ros.org/rosbag
Using MATLAB with ROS bag files
Getting Started with ROS in Simulink

Send messages to `/location` topic

Receive messages sent to `/location` topic
Towards MATLAB Control

We need MATLAB/Simulink to “speak” ROS

- Subscribe to Pose messages and publish Twist

**Goal Waypoint (X, Y)**

MATLAB/Simulink Waypoint Control

**Topic = turtle1/pose**
**Type = turtlesim/Pose**

Turtle Sim

TurtleSim

turtle1/cmd_vel
gometry_msgs/Twist
Message Types

Feedback from “sensors”

`turtlesim/Pose Message`

File: `turtlesim/Pose.msg`

Raw Message Definition

```c
float32 x
float32 y
float32 theta

float32 linear_velocity
float32 angular_velocity
```

Compact Message Definition

```c
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity
```

Command to “actuators”

`geometry_msgs/Twist Message`

File: `geometry_msgs/Twist.msg`

Raw Message Definition

```c
# This expresses velocity in free space broken into its
Vector3 linear
Vector3 angular
```

Compact Message Definition

```c
geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular
```
Waypoint Guidance

Moving to a goal
- Command velocity proportional to distance errors
- Linear Velocity Command
  \[ v^* = K_v \sqrt{(x^* - x)^2 + (y^* - y)^2} \]

- Angular Velocity Command
  - Angle to goal
    \[ \theta^* = \tan^{-1}\left(\frac{y^* - y}{x^* - x}\right) \]
  - Velocity command
  \[ \gamma = K_h (\theta^* \ominus \theta), \ K_h > 0 \]

Considerations
- Angular difference – range [-180, 180)
- Distance threshold, when are we close enough?
**Algorithm** turtle waypoint.m

1. rosinit
2. Define list of waypoints
3. Initialize current waypoint at start of list
4. Create a subscriber to the Pose topic
5. Create a publisher to the Twist topic
6. **while** True **do**
7.   Wait to receive a Pose message
8.   Extract x, y and theta from Pose message
9.   Call waypoint guidance subroutine
10. **if** (Distance to waypoint < threshold) **and** (Still have more waypoints) **then**
11.   Increment current waypoint
12. **else if** Distance to waypoint < threshold **then**
13.   Break from while loop
14. **else**
15.   Publish Twist command from waypoint guidance
16. **end if**
17. **end while**
18. rosshutdown
Video Example

https://vimeo.com/213580986
**Algorithm** Simple Waypoint Guidance - Angle Constraint

**Input:** x, y, theta, WPx, WPy

*Position Error*
1: currentPose = x, y
2: goalPose = WPx, WPy
3: errorPose = || goalPose - currentPose ||

*Angle Error*
4: goalTheta = “angle from current position to WP”
5: currentTheta = theta
6: errorTheta = desiredTheta - theta

*Clamp Angle: ±π*
7: while errorTheta > π do
8:    errorTheta = errorTheta - 2*π
9: end while
10: while errorTheta < -π do
11:    errorTheta = errorTheta + 2*π
12: end while

*Proportional Gain*
13: angularVel = $k_a$ * errorTheta
14: linearVel = $k_l$ * errorPose
15: return

**Output:** distance, linearVel, angularVel
Video Example

https://vimeo.com/214241276
Simulink Implementation

Turtle Waypoint Control

MATLAB Function - Waypoint Guidance

Waypoint Sequence

Real-Time Pacer
Speedup = 1
Simulink Implementation

Turtle Waypoint Control

Waypoint 1
entry: wp_x=1.0; wp_y=1.0;

Waypoint 2
entry: wp_x=10.0; wp_y=1.0;

Waypoint 3
entry: wp_x=10.0; wp_y=10.0;

Waypoint 4
entry: wp_x=1.0; wp_y=10.0;
Wikipedia Example

- States, transitions, conditions
ABV Statechart
Version 5, 10/16/2014

Indicates a "Behavior" which typically has a single end condition, often a timeout.
Indicates a "Task Machine" which can have multiple triggers to determine transitions between behaviors and/or other task machines.
Assignment 3

0. Check your MATLAB installation!

1. Create a launch file
   • Simple XML file with text editor

2. Record and play back data
   • Using rosbag utility
   • Using remap functionality in launch file

3. Plotting rosbag data with MATLAB
   • From a working example, create post-processing script

4. Simulink timing, pacer block
   • Experiment to illustrate sample time of Simulink

5. Turtle feedback control via Simulink
   • Tune working example of waypoint guidance